**Задание 1.**

Реализуйте класс Rectangle для описания прямоугольника, стороны которого параллельны осям координат.

При инициализации экземпляра передаются координаты левой нижней точки прямоугольника x и y, а также его ширина и высота w и h. Таким образом, координаты верхнего правого угла — (x + w) и (y + h).

При вызове метода intersection (например, rect1.intersection(rect2)) должен возвращаться прямоугольник, который возникает как пересечение rect1 и rect2. Если прямоугольники не пересекаются, должен возвращаться объект None.

Также необходимо реализовать метод get для каждого из атрибутов класса, возвращающий заданное значение данного атрибута. Пример: get\_\*() - возвращает значение атрибута \*.

Гарантируется, что во входных данных ширина и высота любого прямоугольника положительны.

Если пересечением прямоугольников является точка или отрезок, то следует считать, что они не пересекаются.

Проверить на правильность можно с помощью кода подобного прилагаемому ниже

rect1 = Rectangle(0, 0, 10, 10)  
rect2 = Rectangle(10, 0, 10, 10)  
rect3 = rect1.intersection(rect2)  
  
**if** rect3 **is None**:  
 print(**'No intersection'**)  
**else**:  
 print(rect3.get\_x(), rect3.get\_y(), rect3.get\_w(), rect3.get\_h())

В данном случае пересечения нет

**Задание 2**

Напишите два класса: LeftParagraph и RightParagraph для печати абзаца с выравниванием по левому и правому краю.

При инициализации экземпляры обоих классов должны принимать целое число — ширину поля вывода. В обоих классах нужно реализовать метод add\_word для добавления слова в абзац и метод end, выводящий полученный абзац на печать и начинающий формирование нового.

Гарантируется, что длина любого слова меньше ширины поля вывода.

Проверить на правильность можно с помощью кода подобного прилагаемому ниже

lp = LeftParagraph(8)  
lp.add\_word(**'abc'**)  
lp.add\_word(**'defg'**)  
lp.add\_word(**'hi'**)  
lp.add\_word(**'jklmnopq'**)  
lp.add\_word(**'r'**)  
lp.add\_word(**'stuv'**)  
lp.end()  
print()  
  
rp = RightParagraph(8)  
rp.add\_word(**'abc'**)  
rp.add\_word(**'defg'**)  
rp.add\_word(**'hi'**)  
rp.add\_word(**'jklmnopq'**)  
rp.add\_word(**'r'**)  
rp.add\_word(**'stuv'**)  
rp.end()  
print()

В результате должно быть выведено

abc defg

hi

jklmnopq

r stuv

abc defg

hi

jklmnopq

r stuv

**Задание 3.**

Ограничение времени 1 секунда

Ограничение памяти 64Mb

Напишите класс, который будет строить карту игры в «Морской бой».

Класс SeaMap должен иметь следующие методы (sm – экземпляр SeaMap):

sm.shoot(row, col, result) — добавить на карту результат выстрела, где

row — индекс ряда карты,

col — индекс вертикальной колонки карты,

result — одна из строк: “miss” (промах), “hit” (попадание), “sink” (потопление корабля).

sm.cell(row, col), который

возвращает ‘.’, если в клетке с координатами row, col может находиться корабль,

возвращает ‘\*’, если в клетку уже стреляли или она находится рядом с потопленным кораблём,

возвращает ‘x’ если в клетке было попадание.

Метод cell будет вызываться, только когда все обнаруженные на карте корабли потоплены.

То есть не нужно помечать ‘\*’ клетки рядом с кораблём, в который попали, но не потопили до конца.

Проверить на правильность можно с помощью кода подобного прилагаемому ниже

sm = SeaMap()  
sm.shoot(2, 0, **'sink'**)  
sm.shoot(6, 9, **'hit'**)  
**for** row **in** range(10):  
 **for** col **in** range(10):  
 print(sm.cell(row, col), end=**''**)  
 print()

В результате должно быть выведено

..........

\*\*........

x\*........

\*\*........

..........

..........

.........x

..........

..........

..........

**Задание 4.**

При разработке компьтерных игр зачастую создаётся очень сложная система классов. В этой задаче Вам будет необходимо сделать небольшую заготовку, которую потом можно будет использовать для создания своей игры.

Реализуйте класс Weapon для оружия, который при инициализации должен принимать имя оружия name, наносимый урон damage, а также радиус, на котором оружие может достать до врага – range.

В классе Weapon должны быть реализованы следующие методы:

hit(actor, target) – удар персонажем actor персонажа target. Метод должен проверять, жив ли персонаж target, если нет, то выводить сообщение «Враг уже повержен», иначе должна проводится проверка расстояния от actor до target. Если расстояние больше range оружия, тогда надо выводить сообщение «Враг слишком далеко для оружия <name>». В случае, если target жив и оружие имеет достаточный range, вывести сообщение «Врагу нанесен урон оружием <name> в размере <damage>» и нанести урон target.

приведение к строке – при приведении к строке должно возвращаться имя оружия.

Реализуйте класс BaseCharacter для создания базовых персонажей, который при инициализации должен принимать pos\_x и pos\_y – позицию персонажа в мире, hp – количество жизней персонажа.

Класс BaseCharacter должен реализовывать методы:

move(delta\_x, delta\_y) – для перемещения персонажа на delta\_x и delta\_y в игровом мире.

is\_alive() – проверка, жив ли персонаж. Возвращает True, если жив, иначе – False.

get\_damage(amount) – убавляет количество жизней персонажа на amount. При достижении 0 или отрицательного значения персонаж считается мёртвым.

get\_coords() – возвращает кортеж с текущими координатами персонажа.

Реализуйте класс BaseEnemy для создания противников, который расширяет класс BaseCharacter. При инициализации он должен принимать pos\_x и pos\_y – позицию персонажа в мире, weapon – оружие, hp – количество жизней персонажа. Класс должен использовать инициализатор BaseCharacter.

Класс BaseEnemy должен реализовывать следующие методы:

hit(target) – для удара персонажа target. Метод должен позволять ударить только персонажа класса MainHero. Для всех других классов должно печататься сообщение «Могу ударить только Главного героя».

приведение к строке – при приведении к строке должно возвращаться «Враг на позиции (<pos\_x>, <pos\_y>) с оружием <weapon>»

Реализуйте класс MainHero для создания главного героя, который расширяет класс BaseCharacter. При инициализации он должен принимать pos\_x и pos\_y – позицию персонажа в мире, name – имя персонажа, hp – количество жизней персонажа. Класс должен использовать инициализатор BaseCharacter.

Класс MainHero должен реализовывать следующие методы:

hit(target) – для удара персонажа target. Метод должен проверять, есть ли у персонажа оружие, если его нет выводить - «Я безоружен», иначе позволять ударить только персонажа класса BaseEnemy. Для всех других классов должно печататься сообщение «Могу ударить только Врага».

add\_weapon(weapon) – добавляет оружие в инвентарь персонажа, если это оружие единственное, то оно должно экипироваться сразу же. Метод должен печатать: «Подобрал <weapon>». Метод должен работать только для переданных объектов класса Weapon, для всего остального он должен печатать сообщение «Это не оружие».

next\_weapon() – для смены оружия. Если оружия нет, то должно выводиться «Я безоружен», если есть только одно оружие – «У меня только одно оружие», иначе должна производиться смена оружия на следующее (в порядке подбора, по кругу) и выводиться сообщение «Сменил оружие на <weapon>».

heal(amount) – для повышения количества жизней персонажа на amount. При этом максимальный уровень здоровья главного героя – 200. Тажже должно выводиться сообщение «Полечился, теперь здоровья <hp>».

Протестировать правильность Ваших классов можно на следующем примере

weapon1 = Weapon(**"Короткий меч"**, 5, 1)  
weapon2 = Weapon(**"Длинный меч"**, 7, 2)  
weapon3 = Weapon(**"Лук"**, 3, 10)  
weapon4 = Weapon(**"Лазерная орбитальная пушка"**, 1000, 1000)  
princess = BaseCharacter(100, 100, 100)  
archer = BaseEnemy(50, 50, weapon3, 100)  
armored\_swordsman = BaseEnemy(10, 10, weapon2, 500)  
archer.hit(armored\_swordsman)  
armored\_swordsman.move(10, 10)  
print(armored\_swordsman.get\_coords())  
main\_hero = MainHero(0, 0, **"Король Артур"**, 200)  
main\_hero.hit(armored\_swordsman)  
main\_hero.next\_weapon()  
main\_hero.add\_weapon(weapon1)  
main\_hero.hit(armored\_swordsman)  
main\_hero.add\_weapon(weapon4)  
main\_hero.hit(armored\_swordsman)  
main\_hero.next\_weapon()  
main\_hero.hit(princess)  
main\_hero.hit(armored\_swordsman)  
main\_hero.hit(armored\_swordsman

Наберите вышеприведенный код после Ваших классов, должно получиться:

Могу ударить только Главного героя

(20, 20)

Я безоружен

Я безоружен

Подобрал Короткий меч

Враг слишком далеко для оружия Короткий меч

Подобрал Лазерная орбитальная пушка

Враг слишком далеко для оружия Короткий меч

Сменил оружие на Лазерная орбитальная пушка

Могу ударить только врага

Врагу нанесен урон оружием Лазерная орбитальная пушка в размере 1000

Враг уже повержен

**Задание 5.**

Реализуйте программу, имитирующую работу почтовых серверов и клиентов.

Клиенты должны иметь возможность принимать и отправлять письма, которые хранятся на серверах.

Вам предстоит спроектировать программу самостоятельно. Чтобы задать направление мысли, давайте обсудим примерный интерфейс почтового клиента.

client = MailClient(server, user) — подключение к учётной записи пользователя user на сервере server.

client.receive\_mail() — получение почты. Когда почта получена, она удаляется с сервера. Что конкретно возвращает этот метод, решите самостоятельно.

client.send\_mail(server1, user1, message) — послать сообщение message пользователю user1 на сервере server1.

Ваша программа должна быть не просто набором классов, но и предоставлять готовый интерфейс, по которому преподаватель (а не автоматическая система тестирования) будет оценивать функциональность и корректность вашей программы.

Программа должна позволять из своего интерфейса определять список серверов, на которые можно отправлять письма. При попытке отправить письмо на сервер, которого нет в списке, программа должна сообщать пользователю, что не может этого сделать.